

# Vitesses d'exécution dans R : comparaisons de base R, dplyr et data.table

Antoine Sireyjol

14 février 2019

# Je me présente

- Antoine Sireyjol, statisticien indépendant

# Je me présente

- Antoine Sireyjol, statisticien indépendant
- Ancien salarié de la Drees : le service statistique du Ministère des Solidarités et de la santé

# Je me présente

- Antoine Sireyjol, statisticien indépendant
- Ancien salarié de la Drees : le service statistique du Ministère des Solidarités et de la santé
- Travail depuis Toulouse sur la migration de leurs outils de SAS vers R (ou Python)

# Je me présente

- Antoine Sireyjol, statisticien indépendant
- Ancien salarié de la Drees : le service statistique du Ministère des Solidarités et de la santé
- Travail depuis Toulouse sur la migration de leurs outils de SAS vers R (ou Python)
- Tests de comparaison d'instructions entre R et SAS mais aussi entre les différentes options de R

# Plan de la présentation

- ① Présentation de dplyr et data.table
  - 1.1. Dplyr
  - 1.2. Data.table
  - 1.3. Comparaisons avec base R

# Plan de la présentation

- ① Présentation de dplyr et data.table
  - 1.1. Dplyr
  - 1.2. Data.table
  - 1.3. Comparaisons avec base R
- ② Comparaison des vitesses d'exécution
  - 2.1. Étude de cas avec le package nycflights13
  - 2.2. Vitesses d'instruction en fonction de la taille de l'échantillon

# Plan de la présentation

- ❶ Présentation de `dplyr` et `data.table`
  - 1.1. `Dplyr`
  - 1.2. `Data.table`
  - 1.3. Comparaisons avec base R
- ❷ Comparaison des vitesses d'exécution
  - 2.1. Étude de cas avec le package `nycflights13`
  - 2.2. Vitesses d'instruction en fonction de la taille de l'échantillon
- ❸ Astuces d'optimisation d'un script R
  - 3.1. Utilisation de `*apply`
  - 3.2. Éviter `ifelse`
  - 3.3. Définition d'une variable à l'intérieur de `summarise`
  - 3.4. `group_by` de `dplyr`



# Plan de la présentation

- ❶ Présentation de `dplyr` et `data.table`
  - 1.1. `Dplyr`
  - 1.2. `Data.table`
  - 1.3. Comparaisons avec base R
- ❷ Comparaison des vitesses d'exécution
  - 2.1. Étude de cas avec le package `nycflights13`
  - 2.2. Vitesses d'instruction en fonction de la taille de l'échantillon
- ❸ Astuces d'optimisation d'un script R
  - 3.1. Utilisation de `*apply`
  - 3.2. Éviter `ifelse`
  - 3.3. Définition d'une variable à l'intérieur de `summarise`
  - 3.4. `group_by` de `dplyr`
- ❹ Conclusions sur les comparaisons

# Plan de la présentation

- ❶ Présentation de `dplyr` et `data.table`
  - 1.1. `Dplyr`
  - 1.2. `Data.table`
  - 1.3. Comparaisons avec base R
- ❷ Comparaison des vitesses d'exécution
  - 2.1. Étude de cas avec le package `nycflights13`
  - 2.2. Vitesses d'instruction en fonction de la taille de l'échantillon
- ❸ Astuces d'optimisation d'un script R
  - 3.1. Utilisation de `*apply`
  - 3.2. Éviter `ifelse`
  - 3.3. Définition d'une variable à l'intérieur de `summarise`
  - 3.4. `group_by` de `dplyr`
- ❹ Conclusions sur les comparaisons
- ❺ Références

# Dplyr et data.table

# Dplyr et le tidyverse

- Tidyverse : environnement d'analyse de données en R

# Dplyr et le tidyverse

- Tidyverse : environnement d'analyse de données en R
- Propre format de données : le tibble

# Dplyr et le tidyverse

- Tidyverse : environnement d'analyse de données en R
- Propre format de données : le tibble
- Syntaxe caractéristique et concurrente des fonctions de base R avec dplyr

# Dplyr et le tidyverse

- Tidyverse : environnement d'analyse de données en R
- Propre format de données : le tibble
- Syntaxe caractéristique et concurrente des fonctions de base R avec dplyr
- Chaînage possible des instructions avec %>%

# Dplyr et le tidyverse

- Tidyverse : environnement d'analyse de données en R
- Propre format de données : le tibble
- Syntaxe caractéristique et concurrente des fonctions de base R avec dplyr
- Chaînage possible des instructions avec %>%
- Très lisible et optimisé



# Syntaxe dplyr (1)

La grammaire dplyr s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables

# Syntaxe dplyr (1)

La grammaire dplyr s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables
- `filter(data, condition)` sélectionne au sein d'une table certaines observations.

# Syntaxe dplyr (1)

La grammaire dplyr s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables
- `filter(data, condition)` sélectionne au sein d'une table certaines observations.
- `arrange(data, var1, descending var2,...)` trie une base selon une ou plusieurs variables.

# Syntaxe dplyr (1)

La grammaire dplyr s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables
- `filter(data, condition)` sélectionne au sein d'une table certaines observations.
- `arrange(data, var1, descending var2,...)` trie une base selon une ou plusieurs variables.
- `select(data, var1 : varX)` sélectionne certaines variables dans une base.

# Syntaxe dplyr (1)

La grammaire dplyr s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables
- `filter(data, condition)` sélectionne au sein d'une table certaines observations.
- `arrange(data, var1, descending var2,...)` trie une base selon une ou plusieurs variables.
- `select(data, var1 : varX)` sélectionne certaines variables dans une base.
- `group_by(data, var)` regroupe une table par une variable

# Syntaxe dplyr (1)

La grammaire dplyr s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables
- `filter(data, condition)` sélectionne au sein d'une table certaines observations.
- `arrange(data, var1, descending var2,...)` trie une base selon une ou plusieurs variables.
- `select(data, var1 : varX)` sélectionne certaines variables dans une base.
- `group_by(data, var)` regroupe une table par une variable
- `summarise(data, newvar1 = mean(var1), newvar2 = sum(var2))` réalise toute sorte d'opérations statistiques sur une table.

## Syntaxe dplyr (2)

- Possibilité de chaîner ces opérations : l'opérateur `%>%`

## Syntaxe dplyr (2)

- Possibilité de chaîner ces opérations : l'opérateur `%>%`
- `fonction(data, params...)` est équivalent à `data %>% fonction(params...)`



## Syntaxe dplyr (2)

- Possibilité de chaîner ces opérations : l'opérateur `%>%`
- `fonction(data, params...)` est équivalent à `data %>%  
fonction(params...)`
- Exemple :

## Syntaxe dplyr (2)

```
library(tidyverse)
# on crée un data frame avec 100 lignes,
# chaque individu appartenant à un des 50 groupes
df <- data.frame(id1 = c(1:100),
                  idgpe = sample(50))

# on y applique les instructions de dplyr
df %>% as_tibble() %>%
  mutate(var = rnorm(100)) %>%
  group_by(idgpe) %>%
  summarise(var_mean = mean(var)) -> output_tibble
print(head(output_tibble), 5)
```

## Syntaxe dplyr (2)

```
## # A tibble: 5 x 2
##   idgpe var_mean
##   <int>   <dbl>
## 1     1     0.840
## 2     2    -0.732
## 3     3    -0.391
## 4     4     1.05
## 5     5    -1.03
```

# Data.table

- Format optimisé de data.frame

# Data.table

- Format optimisé de data.frame
- Complémentaire à base R

# Data.table

- Format optimisé de data.frame
- Complémentaire à base R
- Optimisation de l'opérateur [

# Data.table

- Format optimisé de data.frame
- Complémentaire à base R
- Optimisation de l'opérateur [
- Chaînage possible des instructions

# Data.table

- Format optimisé de data.frame
- Complémentaire à base R
- Optimisation de l'opérateur [
- Chaînage possible des instructions
- Plus lisible, plus rapide que base R



## Syntaxe data.table (1)

- l'opérateur [ appliqué au data.table change de signification et devient :

`DT[i, j, by]`

# Syntaxe data.table (1)

- l'opérateur [ appliqué au data.table change de signification et devient :

`DT[i, j, by]`

- `i` permet de sélectionner des lignes de DT

# Syntaxe data.table (1)

- l'opérateur [ appliqué au data.table change de signification et devient :

`DT[i, j, by]`

- i permet de sélectionner des lignes de DT
- j permet de créer des variables ou d'en sélectionner

## Syntaxe data.table (1)

- l'opérateur [ appliqué au data.table change de signification et devient :

`DT[i, j, by]`

- `i` permet de sélectionner des lignes de DT
- `j` permet de créer des variables ou d'en sélectionner
- `by` permet de regrouper les traitements selon les modalités d'une variable définie

## Syntaxe data.table (1)

- l'opérateur `[` appliqué au `data.table` change de signification et devient :

`DT[i, j, by]`

- `i` permet de sélectionner des lignes de `DT`
- `j` permet de créer des variables ou d'en sélectionner
- `by` permet de regrouper les traitements selon les modalités d'une variable définie
- L'usage de `[` permet de chaîner les opérations :

## Syntaxe data.table (2)

```
library(data.table)
# on convertit notre data frame
# précédemment créé en data.table
dt <- as.data.table(df)

# on y applique les même instructions
dt[, var := rnorm(100)
    ][, list(var_mean = mean(var)),
    by = idgpe] -> output_dt

print(head(output_dt, 5))
```

## Syntaxe data.table (2)

```
##      idgpe    var_mean
## 1:      28  0.06643394
## 2:      23 -0.09844197
## 3:      22 -0.16290008
## 4:      32 -0.32254880
## 5:      37  1.03140067
```

# Comparaisons avec base R

dplyr et data.table présentent un certain nombre d'avantages par rapport à l'usage de base R exclusivement :

- Plus lisibles et moins verbeux, grâce notamment au chaînage



# Comparaisons avec base R

`dplyr` et `data.table` présentent un certain nombre d'avantages par rapport à l'usage de base R exclusivement :

- Plus lisibles et moins verbeux, grâce notamment au chaînage
- Pensés pour l'analyse de données

# Comparaisons avec base R

dplyr et data.table présentent un certain nombre d'avantages par rapport à l'usage de base R exclusivement :

- Plus lisibles et moins verbeux, grâce notamment au chaînage
- Pensés pour l'analyse de données
- Instructions optimisées et bien plus rapides que base R

## Comparaison des vitesses d'exécution

## Étude de cas avec nycflights13

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée

## Étude de cas avec nycflights13

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée
- 336776 lignes et 19 variables

## Étude de cas avec nycflights13

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée
- 336776 lignes et 19 variables
- Base `weather` : indications météo, heure par heure, dans chaque aéroport

# Étude de cas avec nycflights13

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée
- 336776 lignes et 19 variables
- Base `weather` : indications météo, heure par heure, dans chaque aéroport
- 26115 lignes et 15 variables

# Étude de cas avec nycflights13

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée
- 336776 lignes et 19 variables
- Base `weather` : indications météo, heure par heure, dans chaque aéroport
- 26115 lignes et 15 variables
- On crée `flights_dt` et `weather_dt` avec `as.data.table`



# Étude de cas avec nycflights13

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée
- 336776 lignes et 19 variables
- Base `weather` : indications météo, heure par heure, dans chaque aéroport
- 26115 lignes et 15 variables
- On crée `flights_dt` et `weather_dt` avec `as.data.table`
- Étude de cas : fusion des deux tables pour expliquer retards à l'arrivée et au départ en fonction de la météo

# Étude de cas avec nycflights13 - Base R

```
flights_time_hour <- aggregate.data.frame(  
  list(arr_delay = flights$arr_delay,  
        dep_delay = flights$dep_delay),  
  list(time_hour = flights$time_hour,  
        origin = flights$origin),  
  mean)  
merge_base <- merge(weather, flights_time_hour,  
                    by = c("time_hour", "origin"),  
                    sort = FALSE)
```

# Étude de cas avec nycflights13 - dplyr

```
flights %>% group_by(time_hour, origin) %>%  
  summarise(arr_delay = mean(arr_delay),  
            dep_delay = mean(dep_delay)) %>%  
  inner_join(weather, by = c("time_hour", "origin"))  
  ) -> merge_dplyr
```

# Étude de cas avec nycflights13 - data.table

```
merge_DT <- merge(  
  flights_dt[, list(arr_delay = mean(arr_delay),  
                    dep_delay = mean(dep_delay)),  
              by = list(time_hour, origin)],  
  weather_dt,  
  by = c("time_hour", "origin"))
```

# Comparaisons des vitesses de ces instructions

Le package `microbenchmark` nous permet de comparer la vitesse de ces instructions :

```
## Unit: milliseconds
##      expr      min       lq      mean     median
##   base R 647.90814 652.14750 666.51842 661.40331
##   dplyr  49.71914  49.88298  52.48001  50.04483
## data.table 24.78704 25.09367 26.38459 25.15852
##      uq      max neval
## 679.28518 710.27083    10
##  50.99174  62.47419    10
##  25.37925  33.75044    10
```

# Comparaisons du groupage en fonction du nombre d'observations

- Avantage net de dplyr et data.table sur base R, et avantage à data.table sur cet exemple

# Comparaisons du groupage en fonction du nombre d'observations

- Avantage net de dplyr et data.table sur base R, et avantage à data.table sur cet exemple
- Qu'en est-il quand on fait varier le nombre d'observations?

# Comparaisons du groupage en fonction du nombre d'observations

- Avantage net de dplyr et data.table sur base R, et avantage à data.table sur cet exemple
- Qu'en est-il quand on fait varier le nombre d'observations?
- Comparaisons des vitesses d'agrégation en faisant varier le nombre d'observations et le nombre de groupes



# Comparaisons du groupage en fonction du nombre d'observations

- Avantage net de dplyr et data.table sur base R, et avantage à data.table sur cet exemple
- Qu'en est-il quand on fait varier le nombre d'observations?
- Comparaisons des vitesses d'agrégation en faisant varier le nombre d'observations et le nombre de groupes
- Les instructions testées :

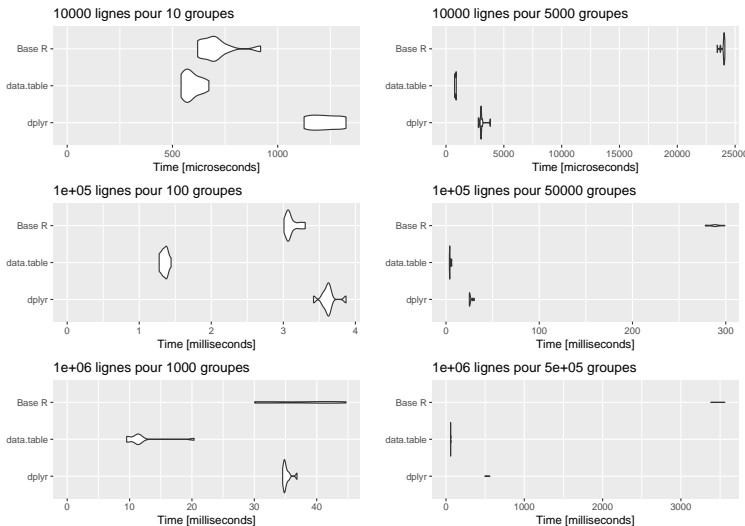
# Comparaisons du groupage en fonction du nombre d'observations

```
# Pour dplyr
datatib %>% group_by(y)
%>% summarise(x = mean(x))

# Pour data.table
dataDT[, .(x = mean(x)),
        by = .(y = y)]

# Pour base R
tap <- tapply(test$x, test$y, mean)
data.frame(x = tap, y = names(tap))
```

# Résultats en fonction des nombres de lignes et de groupes



## Quelques astuces d'optimisation

# Utiliser les fonctions \*apply plutôt que les boucles

- Important de vectoriser ses instructions

# Utiliser les fonctions \*apply plutôt que les boucles

- Important de vectoriser ses instructions
- \*apply permet d'appliquer une fonction à un ensemble d'éléments

# Utiliser les fonctions `*apply` plutôt que les boucles

- Important de vectoriser ses instructions
- `*apply` permet d'appliquer une fonction à un ensemble d'éléments
- `apply(matrice, i, f())` applique `f` à l'ensemble des lignes ( $i = 1$ ) ou des colonnes ( $i = 2$ ) de la matrice

# Utiliser les fonctions \*apply plutôt que les boucles

- Important de vectoriser ses instructions
- \*apply permet d'appliquer une fonction à un ensemble d'éléments
- `apply(matrice, i, f())` applique `f` à l'ensemble des lignes ( $i = 1$ ) ou des colonnes ( $i = 2$ ) de la matrice
- `lapply(X, f())` applique `f` à chacun des éléments du vecteur ou de la liste `X`



# Utiliser les fonctions \*apply plutôt que les boucles

- Important de vectoriser ses instructions
- \*apply permet d'appliquer une fonction à un ensemble d'éléments
- `apply(matrice, i, f())` applique `f` à l'ensemble des lignes ( $i = 1$ ) ou des colonnes ( $i = 2$ ) de la matrice
- `lapply(X, f())` applique `f` à chacun des éléments du vecteur ou de la liste `X`
- Comparaisons avec une boucle :

# lapply et boucle

```
# On crée une matrice de 10 000 lignes et 25 colonnes
data <- matrix(1:250000, ncol = 25)

# On veut le résultat de la somme de chaque ligne
# Avec une boucle
boucle_results <- c()
for (i in 1:nrow(data)){
  boucle_results <- append(boucle_results, sum(data[i, ]))
}

# Avec apply
apply_results <- apply(data, 1, sum)

identical(apply_results, boucle_results)
```

```
## [1] TRUE
```

## lapply et boucle : microbenchmark

```
## Unit: milliseconds
##      expr      min       lq      mean     median       uq
##  boucle 112.24680 113.59336 114.22907 113.94664 114.48282
##   apply  10.17002  10.26844  10.43512  10.42659  10.54435
##           max neval
## 117.37590     20
##  10.86463     20
```

# Éviter ifelse

```
# Fonction ifelse
flights$gros_retard <- ifelse(flights$arr_delay > 30,
                             "oui", "non")

# Sans ifelse
flights$gros_retardbis <- "non"
flights$gros_retardbis[flights$arr_delay > 30] <- "oui"
```

# Éviter ifelse - microbenchmark

```
## Unit: milliseconds
##      expr      min       lq     mean   median
##      ifelse 75.796985 76.086833 81.54956 77.468095
## sans ifelse  3.351323  3.379198  3.91364  3.409349
##      uq      max neval
## 85.958470 159.76776   100
##  3.495252  16.71053   100
```

## Dplyr : pas de création de variable à l'intérieur de summarise()

*# Avec mutate*

```
flights %>% mutate(propor_delay = arr_delay / air_time) %>%  
group_by(time_hour) %>%  
summarise(propor_delay = mean(propor_delay)  
          ) -> output_dyp
```

*# Sans mutate*

```
flights %>% group_by(time_hour) %>%  
summarise(propor_delay = mean(arr_delay / air_time)  
          ) -> output_dyp2
```

## Dplyr : pas de création de variable à l'intérieur de summarise() - microbenchmark

```
## Unit: milliseconds
##           expr           min           lq           mean           median
##   dplyr_mutate   21.57225   21.7139   22.6809   22.28478
## dplyr_sans_mutate 213.50097 215.4244 218.2662 217.39672
##           uq           max neval
##   23.16684 26.2542     10
## 222.01496 224.6745     10
```

## Dplyr : group\_by par factor plutôt que caractère

```
flights$originfac <- as.factor(flights$origin)

# group by character
flights %>% group_by(origin) %>%
  summarize(mean_delay = mean(arr_delay, na.rm = TRUE)
    ) -> out_char

# group by factor
flights %>% group_by(originfac) %>%
  summarize(mean_delay = mean(arr_delay, na.rm = TRUE)
    ) -> out_fact
```



# Dplyr : group\_by par factor plutôt que caractère - microbenchmark

```
## Unit: milliseconds
##           expr           min           lq          mean        median
## group by character 13.26932 13.89624 15.37959 14.51007
##   group by factor 10.75028 11.50918 12.14873 11.72337
##           uq           max neval
## 14.96575 24.76030      20
## 11.88294 21.66953      20
```

## Conclusions sur les comparaisons

# Conclusions

- Dplyr et data.table : + rapides et + faciles d'utilisation que base R pour l'analyse de données

# Conclusions

- Dplyr et data.table : + rapides et + faciles d'utilisation que base R pour l'analyse de données
- Choix entre dplyr et data.table dépend de différents facteurs :

# Conclusions

- Dplyr et data.table : + rapides et + faciles d'utilisation que base R pour l'analyse de données
- Choix entre dplyr et data.table dépend de différents facteurs :
- Type de base de données en entrée

# Conclusions

- Dplyr et data.table : + rapides et + faciles d'utilisation que base R pour l'analyse de données
- Choix entre dplyr et data.table dépend de différents facteurs :
- Type de base de données en entrée
- Profil des personnes qui codent

# Conclusions

- Dplyr et data.table : + rapides et + faciles d'utilisation que base R pour l'analyse de données
- Choix entre dplyr et data.table dépend de différents facteurs :
- Type de base de données en entrée
- Profil des personnes qui codent
- Importance accordée à la vitesse d'exécution

# Conclusions

- Dplyr et data.table : + rapides et + faciles d'utilisation que base R pour l'analyse de données
- Choix entre dplyr et data.table dépend de différents facteurs :
- Type de base de données en entrée
- Profil des personnes qui codent
- Importance accordée à la vitesse d'exécution
- Intéressant de faire des tests sur ses scripts pour voir comment ils peuvent être optimisés



## Références

# Références

- Formation R perfectionnement, M. Chevalier

# Références

- Formation R perfectionnement, M. Chevalier
- Introduction à R et au tidyverse, J.Barnier

# Références

- Formation R perfectionnement, M. Chevalier
- Introduction à R et au tidyverse, J. Barnier
- Manipulations avancées avec data.table, J. Larmarange

# Références

- Formation R perfectionnement, M. Chevalier
- Introduction à R et au tidyverse, J. Barnier
- Manipulations avancées avec data.table, J. Larmarange
- Pour des benchmarks data.table, dplyr et python : Benchmarks : Grouping

# Références

- Formation R perfectionnement, M. Chevalier
- Introduction à R et au tidyverse, J. Barnier
- Manipulations avancées avec data.table, J. Larmarange
- Pour des benchmarks data.table, dplyr et python : Benchmarks : Grouping
- <https://antoinesir.rbind.io>